

Task Assignment on Spatial Crowdsourcing (Technical Report)

Peng Cheng, Xun Jian, Lei Chen

Hong Kong University of Science and Technology, Hong Kong, China
 {pchengaa, leichen}@cse.ust.hk, xjian@connect.ust.hk

ABSTRACT

Recently, with the rapid development of mobile devices and the crowdsourcing platforms, the spatial crowdsourcing has attracted much attention from the database community. Specifically, spatial crowdsourcing refers to sending a location-based request to workers according to their positions, and workers need to physically move to specified locations to conduct tasks. Many works have studied task assignment problems in spatial crowdsourcing, however, their problem definitions are quite different from each other. As a result, there is no work to compare the performances of existing algorithms on task assignment in spatial crowdsourcing. In this paper, we present a comprehensive experimental comparison of most existing algorithms on task assignment in spatial crowdsourcing. Specifically, we first give some general definitions about spatial workers and spatial tasks based on definitions in the existing works studying task assignment problems in spatial crowdsourcing such that the existing algorithms can be applied on same synthetic and real data sets. Then, we provide a uniform implementation for all the algorithms of task assignment problems in spatial crowdsourcing. Finally, based on the results on both synthetic and real data sets, we conclude the strengths and weaknesses of tested algorithms, which can guide further researches on the same area and practical implementations of spatial crowdsourcing systems.

1. INTRODUCTION

With the ubiquity of smart devices equipped with various sensors (e.g., GPS) and the convenience of wireless mobile networks (e.g., 5G), nowadays people can easily participate in *spatial tasks* requiring to be conducted at specified locations that are close to their current locations, such as taking photos/videos, measuring noise levels, delivering packages, and/or reporting waiting times of hot restaurants. As a result, a new framework, spatial crowdsourcing, which enables workers to conduct spatial tasks, has emerged in both academia (e.g., the database community) and industry (e.g., Waze). In general, a spatial crowdsourcing system (e.g., gMission [4, 1] and MediaQ [17]) can assign or recommend spatial tasks to the active workers close to the required locations. The task assignment or task recommendation are based on the current locations of

workers and tasks [15, 10, 23], and/or the specific requirements of tasks and the profiles of workers [6, 7].

The most significant difference between the spatial crowdsourcing and the ordinary crowdsourcing is that the workers can only conduct a part of spatial tasks close enough to them such that the workers can physically move to the required locations before the deadlines of tasks in spatial crowdsourcing problems while the workers have no spatial constraints in ordinary crowdsourcing problems. Thus, studying and designing an effective strategy to help workers conduct spatial tasks located in proximity positions is the major goal for the most existing works on spatial crowdsourcing [15, 16, 23, 10, 19, 6, 7, 12, 21, 13].

In [15], tasks on spatial crowdsourcing platforms can be published in two different modes: *Worker Selected Tasks* (WST) and *Server Assigned Tasks* (SAT). For a system in WST mode [10, 19], online workers can select any spatial tasks posted by the requesters in vicinity by themselves without coordination with the spatial crowdsourcing server. As the workers can choose tasks autonomously, the server has no control on the workers to achieve a global optimal assignment strategy in terms of number of assigned tasks or some objective score. The existing works [10, 19] in WST mode are focusing on planning an optimal route for a particular worker such that the number of tasks can be performed on the route is maximized. On a platform in SAT mode [15, 16, 23, 6, 7], the online workers will report their locations to the server periodically, then the server has a global picture of the tasks and workers such that it can assign tasks to nearby workers to achieve the global optimum in terms of some objective functions. In WST mode, workers do not need to keep reporting their locations to the server while the server needs to tracking the workers' positions in SAT mode such that the spatial privacy of workers needs to be protected. [14]. Moreover, In WST mode, as the server has no control on the workers, some tasks may never be selected. However, in SAT mode, the server may assign these tasks to workers to maximize the number of assigned tasks.

Although most researches on spatial crowdsourcing study tasks assignment problems, their problem settings and optimization objectives are not quite same. In [15], each worker has a spatial working area and can only accept the tasks within the working area. In [6], workers has no working area limitations and the platform will pay the traveling cost when assign a worker to move to another location. In [15, 16, 23], the optimization goal is to maximize the number of the assigned tasks for the entire platform. In [10, 19], the optimization goal is to maximize the number of planed tasks for a particular worker who can conduct them one-by-one. The work [7] focuses on maximizing the reliability and diversity of finished tasks. There have been many works studying task assignment on spatial crowdsourcing, however, no work has compared the existing

algorithms tailed for different settings. Currently, there is no uniform baseline implementations, thus the results in different works cannot be compared directly as the differences in detailed implementations may blur the contributions of algorithms. Thus, it is essential to provide a fair comparison study over the existing algorithms on a common spatial crowdsourcing platform to show their pros and cons. To compare the existing works in spatial crowdsourcing, we need to set up a common suitable experiment setting with same implementation skills for all the methods we studied. In addition, we will show the performances of the methods on important spatial crowdsourcing metrics, such as running times, numbers of finished tasks, average moving distances and average workloads of workers. As a result, the uniform baseline implementation can avoid the "noises" from implementation skills (e.g., Java V.S. C++), settings and metrics, then report the true contributions of algorithms.

On a spatial crowdsourcing platform, tasks are keeping coming and finishing, and workers are keeping arriving and free to leave. The platforms have no information about the future arrival tasks and workers, then the existing works [15, 16, 23, 6, 10, 19] always try to achieve an optimal assignment for each timestamp at the current situation, which is in fact an approximately optimal result for the entire span of time. Although in some works [15, 23], the assignment problem is reduced to the maximum flow problem, which can be solved in polynomial time, the results for the entire time span are still approximated. When evaluating task assignment algorithms, we compare the approximately optimal results on the entire span of time. In addition, currently, there is no highly customizable spatial crowdsourcing platforms to run comparison experiments for all the existing spatial crowdsourcing algorithms. We utilize an open source simulation tool to generate data sets either following given distributions (e.g., Normal distribution, Zipf distribution, Skewed distribution and Uniform distribution) or based on real spatial/temporal data sets (e.g., Gowalla and Twitter), which can help to compare algorithms with different parameters more accurately.

We find WST algorithms in fact can perform well on real dataset, although they have no global pictures. Especially, progressive algorithm (PRS), an algorithm in WST mode, can finish more tasks than other tested approaches and runs fast. Moreover, the confidences of tasks need to be particularly concerned when the required confidences of tasks are high and/or the reliabilities of workers are low. Other detailed findings can be found in Section 5.3.

To summarize, we try to make the following contributions:

- We propose a uniform definition for task assignment problems in both SAT and WST modes in Section 2, which adopts most existing works in this area and can be a footstone for the future studies in this area.
- We provide uniform baseline implementations for all existing algorithms in both SAT mode and WST mode. These implementations adopt common basic operations and offer a base for comparing with future works in this area.
- We propose an objective and sufficient experimental evaluation and test the performances of the existing algorithms over extensive benchmarks in Section 5.
- We conclude the advantages and disadvantages of publishing tasks in either SAT mode or WST mode based on the results of our experimental evaluation in Section 5.3.

Section 3 and Section 4 introduce existing algorithms in SAT mode and WST mode respectively. Section 6 concludes this paper.

2. UNIFORM DEFINITION

In this section, we give a uniform definition about task assignment in spatial crowdsourcing, which is based on the definitions in existing works [15, 16, 10, 7].

Definition 1. (*Workers*) Let $W_p = \{w_1, w_2, \dots, w_n\}$ be a set of n workers at timestamp p . Each worker w_i ($1 \leq i \leq n$) is located at position $l_i(p)$ at timestamp p , can move with velocity v_i , specifies a working area a_i , has a reliability value r_i and a capacity value c_i . ■

In Definition 1, worker w_i can move dynamically with speed v_i in any direction, and at each timestamp p , he/she is located at spatial places $l_i(p)$, and prefer to conduct the tasks located in their working areas. The working area a_i can be any shapes. Here we define it as a square area whose center is at the spatial place $l_i(p)$ and side length is s_i . Based on the history performance of each worker, we can have his/her reliability value r_i , which indicates the possibility he/she can finish the assigned task. Moreover, each worker may accept at most c_i tasks at the same time and conducts the assigned tasks one by one. The workers can freely join or leave the spatial crowdsourcing system. When a worker w_i joins the system, he/she can be either available or busy. Here being available means the worker can be assigned more tasks while being busy indicates the number of assigned tasks of worker w_i has reached the capacity of him/her c_i and no more tasks can be assigned unless he/she finishes or rejects some assigned tasks.

Definition 2. (*Spatial Tasks*) Let $T_p = \{t_1, t_2, \dots, t_m\}$ be a set of time-constrained spatial tasks at timestamp p . Each task t_j ($1 \leq j \leq m$) is published at timestamp s_j and located at a specific location l_j , then workers are expected to reach the location of task t_j before the arrival deadline e_j . To guarantee the quality, each task may require c_j answers and specify an expected quality level q_j . ■

As given in Definition 2, usually, a task requester creates a time-constrained spatial task t_j at timestamp s_j , which requires workers to physically move to a specific location l_j and arriving at l_j before the arrival deadline e_j . To accomplish a task t_j , the system needs to collect c_j (odd integer) answers from workers to guarantee the expected quality of task t_j is at least q_j . In order to aggregate the answers from c_j workers, we use majority voting, which reports the same answer of the majority workers (not less than $\frac{c_j+1}{2}$ workers) as the result of t_j . In fact, we can use any aggregation methods to report the final results, however, to compare the methods in spatial crowdsourcing, we follow majority voting method in exiting work [16]. Let the set W_j be the workers that answer task t_j . We can compute the accuracy of a task as below

$$\Pr(W_j) = \sum_{x=\frac{c_j+1}{2}}^{c_j} \sum_{W_{j,x}} \left(\prod_{w_i \in W_{j,x}} r_{ij} \prod_{w_i \in W_j - W_{j,x}} (1 - r_{ij}) \right), \quad (1)$$

where $W_{j,x}$ indicates the subsets with x workers of the entire worker set W_j who answers task t_j .

Definition 3. (*Task Assignment Instance Set*) At timestamp p , given a worker set W_p and a task set T_p , a task assignment instance set, denoted by I_p , is a set of worker-and-task assignment pairs in the form $\langle w_i, t_j \rangle$, where a spatial task t_j is assigned to a worker w_i while satisfying the constraints of workers and tasks. ■

Here, each worker-and-task pair $\langle w_i, t_j \rangle$ in I_p indicates the required location l_j of task t_j is in the working area of worker w_i

and he/she can reach l_j before the arrival deadline e_j . Moreover, the capacity constraint of worker w_i is satisfied, which means the number of assigned tasks for worker w_i is not larger than his/her capacity c_i .

3. ALGORITHMS IN SERVER-ASSIGNED-TASKS MODE

In this section, we will introduce four typical algorithms for task assignment on spatial crowdsourcing in SAT mode. They can be categorized into two groups from the perspective of the number of required answers of each task. The first group just includes one algorithm, maximum flow based algorithm (G-greedy) [15, 23], which assume the reliability of each workers is 100 %, then the problem can be reduced to maximum flow problem and the existing algorithms for maximum flow problem can be used to solve this kind of task assignment problem. However, as discussed in Section 1 the spatial crowdsourcing platforms are highly dynamic, for each timestamp maximum flow based algorithms can exactly solve the problem instance while for the entire time span the result is in fact a local optimal result. In order to improve the result for the entire time span, least location entropy priority algorithm (G-llep) and nearest neighbor priority algorithm (G-nnp) are introduced based on G-greedy. The second group includes sampling based algorithm (RDB-sam) [7], divide-and-conquer based algorithm (RDB-g&c) [6, 7] and heuristic-enhanced greedy algorithm (GT-hgr) [16], which take the reliability and the quality results of tasks into consideration, where each worker has a reliability value, each task requires a quality level for the minimum expected quality value, and the task assignment problem is proved to be a NP-hard problem.

3.1 Maximum Flow Based Algorithms

In this section, we summarize two heuristic algorithms which are evolved from the maximum flow algorithm and target on maximizing the total number of assigned tasks during the time interval ϕ . We first represent the reduction to the maximum flow problem, then introduce the two heuristic algorithms.

3.1.1 Reduction to Maximum Flow Problem

When the workers are assumed to be 100% reliable, the problem to maximize the number of assigned tasks at each timestamp can be reduced to the maximum flow problem.

For a given time instance p with a set of online workers W_p and a set of available tasks T_p , we can create a flow network graph $G_p = (V, E)$ with V as the set of vertices, and E as the set of edges at timestamp p . The set V contains $|W_p| + |T_p| + 2$ vertices. Each worker w_i maps to a vertex w_i and each task maps to a vertex t_j in graph G_p . In addition, we create a *src* vertex and a *dest* vertex. We first connect *src* vertex and every worker vertex w_i and set the capacity for each of these edges as the capacity c_i of worker w_i since each worker can buffer at most c_i tasks. Each task vertex t_j is linked to the *dest* vertex and the capacity is set to 1, as each task only needs one worker to perform. What is more, as each worker w_i can only accept the tasks located inside his/her working area a_i , for every worker vertex w_i we add edges to all the tasks vertices that the corresponding tasks are inside the spatial working area a_i , and set the capacity of each edge to 1.

Figure 1 illustrates an example of this reduction. In Figure 1(a), it is an example of a set of workers W_p and a set of tasks T_p at time instance p . Each worker w_i has a capacity value c_i and a working area shown as a square a_i around the worker. At the same time, Figure 1(b) shows the created maximum flow network graph. One

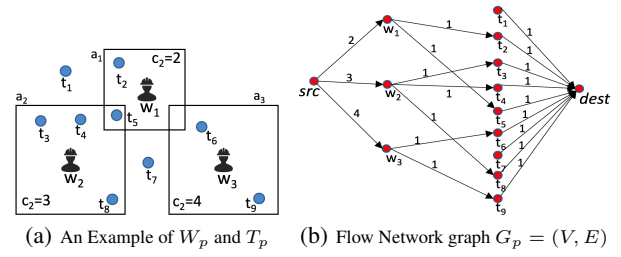


Figure 1: An Example of the Reduction of Maximum Flow Problem.

link from worker vertex w_i to task vertex t_j exists only when the task t_j is located inside the working area a_i of worker w_i . For example, worker vertex w_3 is connected to task vertex t_6 as task t_6 is located inside the working area a_2 of worker w_2 .

With the reduction to maximum flow problem, existing maximum flow algorithms can be used to solve these task assignment problem for a given timestamp. The Ford-Fulkerson algorithm [18] is one well-known algorithm to computing the maximum flow. Note that, this reduction does not necessarily result in a globally optimal answer for the entire time span ϕ . Two algorithms with heuristics are designed to improve the result obtained with only maximum flow algorithms.

3.1.2 Least Location Entropy Priority Algorithm

The first maximum flow-based heuristic algorithm is least location entropy priority algorithm (G-llep) [15], which gives higher priority to the tasks located in worker-sparse areas (areas with low workers densities). The intuition of this algorithm is that for a task located in worker-sparse areas, it is less likely that the task can have a potential worker to select in future timestamps, as less workers will appear in these areas in the future. Thus, for the tasks located in worker-sparse areas, the algorithm tries to assign them workers first by giving them higher priorities at the current timestamp when they are competing with tasks located in worker-dense areas. In other words, if a task located in worker-dense area is not assigned to any workers at the current timestamp, it has a higher possibility to be assigned to some other worker in the future timestamps compared with tasks in worker-sparse areas.

The algorithm utilizes *location entropy* [9] to measure the total number of workers in a location as well as the relative proportion of their future visits to that location. A location with high location entropy indicates many workers visit that location with equal proportions. In other words, for a given location, if only a small number of workers often visit it, its location entropy is low.

For a given location l , let O_l be the set of visits to it, W_l be the set of distinct workers that visited l , and $O_{w,l}$ be the set of visits belonging to worker w . Then, the location entropy for l is calculated as follows:

$$Entropy(l) = - \sum_{w \in W_l} P_l(w) \cdot \log P_l(w), \quad (2)$$

where $P_l(w) = \frac{|O_{w,l}|}{|O_l|}$ is the fraction of total visits to l made by worker w . For each location l , the entropy of it can be treated as its cost value, then the optimization goal of this algorithm is to assign as many tasks as possible with minimum total cost associated to the assigned tasks in each timestamp, which can be reduced to the minimum-cost maximum flow problem [3].

To solve the minimum-cost maximum flow problem, one of the well-known techniques [3] is to first find the maximum flow of the network, then use linear programming method to minimize the

total cost of the flow. Let $G_p = (V, E)$ be the flow network graph for timestamp p . For each edge $(u, v) \in E$, the capacity is $c(u, v) > 0$, the flow $f(u, v) \geq 0$, and the cost is $a(u, v) \geq 0$. The cost of sending the flow $f(u, v)$ is $f(u, v) \cdot a(u, v)$. Denote the maximum flow sent from src vertex to $dest$ vertex as f_{max} , then the linear programming to minimize the total cost can be represented as below:

$$\begin{aligned}
& \text{minimize} && \sum_{(u,v) \in E} f(u, v) \cdot a(u, v) \\
& \text{s.t.} && f(u, v) \leq c(u, v), \\
& && f(u, v) = -f(v, u), \\
& && \sum_{w \in V} f(u, w) = 0 \text{ for all } u \neq src, dest \\
& && \sum_{w \in V} f(src, w) = f_{max} \text{ and } \sum_{w \in V} f(w, dest) = f_{max}
\end{aligned}$$

Note that, this algorithm is not to solve multi-objective optimization problem. In fact, G-llep maximizes the number of assigned tasks first, then minimizes the total cost guaranteeing that the total number of assigned tasks keeps maximized. In other words, maximizing the total number of assigned tasks is the primary objective.

3.1.3 Nearest Neighbor Priority Algorithm

In spatial crowdsourcing, workers need to physically move to specific locations required by requesters in order to conduct spatial tasks. Although the working area constraint can prevent the workers from moving too far away, if the moving distances can be further reduced, the overall assignment process for the entire time span will be improved, as workers can finish assigned tasks faster with shorter moving distances, which in fact improves the efficiency of workers.

Different from only maximizing the total number of assigned tasks for the entire time span in G-greedy and G-llep, nearest neighbor priority algorithm (G-nnp) [15] takes the moving distances of workers into consideration. For G-nnp, the optimization goal becomes maximizing the number of assigned tasks first, then minimizing the total moving distance of workers. G-nnp defines the *travel cost* of worker w to task t as the Euclidean distance between them, denoted as $d(w, t)$. In the network flow graph, each edge between a worker vertex and a task vertex is associated with a cost as the travel cost of the worker to the task. Then the problem turns into the minimum-cost maximum flow problem and the technique in Section 3.1.2 with a different cost function can be applied on it.

In real systems, workers may make mistakes or submit wrong answers deliberately such that the received answers are not totally reliable. To guarantee the reliability of tasks, existing works model the reliability values of workers and try to maximize the expected reliability of tasks. In the rest part of this section, we will introduce 3 algorithms which take into the consideration of reliabilities of workers and tasks.

3.2 Sampling-Based Algorithm

The sampling algorithm (RDB-sam) is proposed to solve reliable diversity based spatial crowdsourcing problem (RDB-SC) [7], which assumes each worker is associated with a reliability value and each task can be estimated with a reliability value. RDB-SC problem tries to obtain a worker-and-task assignment strategy such that the reliability-and-diversity score is maximized. Each worker-and-task assignment strategy is a worker-and-task matching result

and tell which worker should conduct which task at each timestamp. RDB-SC is proved to be NP-hard, thus not tractable. RDB-sam, as an approximation algorithm, can achieve a worker-and-task assignment strategy with high reliable-and-diversity score on the fly. We generally introduce the algorithm as follows. The algorithm first estimates the number of sample size k , where each sample is a possible assignment strategy. Then, it randomly generates k different sampled assignment strategies and report the one with the highest reliable-and-diversity score as the final result.

Note that, RDB-SC problem is a dual-objective optimization problem involving two objective functions to be optimized simultaneously, thus the assignment strategies cannot be ordered directly. In the case, no sample dominates all other samples, the algorithm select one sample with the highest ranking score (i.e., dominating the most number of other samples) [25].

The most important and difficult part of this algorithm is the estimation of the sample size K . The algorithm provides a method to estimate a sample size K such that the “best” sample among the K samples can achieve a (ϵ, δ) -bound, which means the “best” sample is within top ϵ of the entire population with probability δ . To find the value K , the algorithm conducts a binary search within $\left(\frac{p \cdot M \cdot e - 1 + p}{1 - p + e \cdot p}, M\right]$, such that \hat{K} is the smallest K value such that $Pr\{X \leq (1 - \epsilon) \cdot N\} \leq 1 - \delta$ (variable X be the rank of the largest sample, S_K , in the entire population and N is the size of the entire population), where $p = \prod_{j=1}^n \frac{1}{deg(w_j)}$, $M = (1 - \epsilon) \cdot N$, and e is the base of the natural logarithm.

3.3 Divide-and-Conquer-Based Algorithms

When the studied problems are proved to be NP-hard, the computation complexity always increases dramatically with the increase of problem space. However, we can divide the whole problem instance into several subproblem instances with smaller problem spaces, solve the subproblems instances, and merge the results of subproblem instances meanwhile sacrificing minimum accuracies, then it can make a trade-off between accuracy and running time. Inspired by this intuition, divide-and-conquer algorithm (RDB-g&c) [7] and g -divide-and-conquer algorithm (g -D&C) [6] are proposed under the framework of divide-and-conquer algorithms.

Same with RDB-sam, RDB-g&c is proposed for RDB-SC problem. However, g -D&C is designed for *multi-skill oriented spatial crowdsourcing problem* (MS-SC), which assumes each task is associated with a set of specified skills and a budget for traveling compensation and each worker is associated with a set of skills that he/she are good at. MS-SC tries to find an optimal worker-and-task assignment strategy, such that skills provided by workers can cover the skills specified by finished tasks, and workers’ benefits are maximized under the budget constraint. When dividing original problem instance, RDB-g&c keeps dividing the current problem instance into two subproblem instances according to the K-Means clustering result on the locations of tasks whereas g -D&C keeps dividing the current problem into g subproblem instances as shown in Figure 2, where g are determined by a cost model to minimize the running time of the g -D&C algorithm. In detail, g -D&C iteratively choose anchor tasks with a sweeping style (always choosing the task with smallest longitude as the next anchor task), and then each anchor with its top- $(\lceil m/g \rceil - 1)$ nearest tasks organize as the task set of a subproblem instance. The worker set of each subproblem in both RDB-g&c and g -D&C is consisted of all the workers that are valid to any task in the task set of the subproblem.

Any worker may exist in more than one subproblems and also can be assigned to tasks exceeding the capacity of the worker in divide-and-conquer-based algorithms, as the algorithms solve each

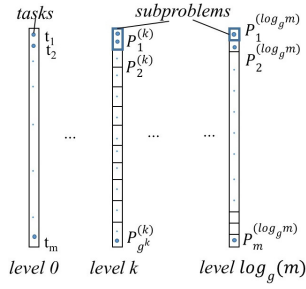


Figure 2: Illustration of the g -D&C.

subproblem without coordinating with other subproblems, which may violate the capacity constraint of workers. When the divide-and-conquer-based algorithms are merging the results of subproblems, worker conflicts may exist between them. To reconcile the conflicts, the algorithms first estimate the cost (e.g., the reliable-and-diversity score in RDB-SC and the flexible budget in MS-SC) of replacing the worker in each subproblem, then they greedily substitute the worker having lower replacing cost with the “best” available worker in the current situation. Here, the “best” available worker is the worker who can improve the reliable-and-diversity score in RDB-SC or the flexible budget in MS-SC for the target task most.

What is more, divide-and-conquer-based algorithms are a good framework to solve complex spatial crowdsourcing problems, which indicates that even the task number is small (e.g., only one task exists), the problem is still NP-hard. For these complex spatial crowdsourcing problems, when the number of tasks is small, existing heuristic or approximation algorithms can achieve a better local optimal result compared with the results of problems with more tasks. When merging the results of subproblems, the conflicts reconciling algorithm tries to maintain these better local optimal results. As a result, divide-and-conquer algorithms can always achieve better results compared with the global algorithm (e.g., the greedy algorithm). On the other hand, if the conflicts between subproblems happens frequently, the time cost of reconciling conflicts will be large and the performance the local optimal maintaining may decrease. Thus, the trade-off between accuracy and running time will be less effective when the worker conflicts between subproblems frequently emerge.

3.4 Heuristic-Enhanced Greedy Algorithm

When considering the accuracies of workers, more than one workers will be assigned to a same task, then usually some aggregation methods can merge the answers from different workers and report a final result. Heuristic-Enhanced Greedy Algorithm (GT-hgr) [16] is proposed under this assumption. In [16], they assume each worker has a reputation score indicating the accuracy of him/her and each task specifies a confidence score, then only when the aggregate reputation score of a task t_i is higher than its specified confidence score α_i , the task is treated as a finished task. For a given task t_i and its assigned workers Q , its aggregate reputation score ($ARS(t_i)$) is the probability that at least $\frac{|Q|+1}{2}$ number of the workers perform the task t correctly, which is calculated as below,

$$ARS(t_i) = \sum_{k=\frac{|Q|+1}{2}}^{|Q|} \sum_{A \in F_k} \prod_{w_j \in A} r_i \prod_{w_j \notin A} (1 - r_i), \quad (3)$$

where F_k is all the subsets of Q with size k , and r_j is the reputation of the worker w_j .

GT-hgr is proposed to solve Maximum Correct Task Assignment problem (MCTA), whose optimization goal is to maximize the number of correctly assigned tasks. For each correctly assigned task t_i , its aggregate reputation score $ARS(t_i)$ is not smaller than its confidence score α_i (i.e., $ARS(t_i) \geq \alpha_i$), and other constraints (e.g, spatial regions of workers, and maximum number of acceptable tasks for each worker) of the task and its assigned workers are satisfied. The MCTA problem is proved NP-hard by reducing from maximum 3-dimensional matching problem (M3M) [20].

GT-hgr utilizes three heuristics to improve the result of a basic greedy algorithm (GT-greedy), which greedily assign a task to one correct match until no further tasks can be assigned. Here one correct match is a task-and-workers pair $\langle t_i, C \rangle$ (C is a set of workers) whose aggregate reputation score $ARS(C)$ is not less than the confidence level α_i of task t_i . The first heuristic is filtering heuristic, which can reduce the size of correct matches by pruning the dominated correct matches. For two correct matches $\langle t_i, C \rangle$ and $\langle t_i, C' \rangle$, if $C \subseteq C'$, match $\langle t_i, C \rangle$ dominates $\langle t_i, C' \rangle$. The second heuristic is least worker assigned (LWA) heuristic, which associates a higher priority for matches with less workers. The last heuristic is least aggregate distance (LAD) heuristic, which prefers the match with smaller summation of moving distances of the workers in that match. GT-hgr first removes dominated correct matches to reduce the problem space. Then, it orders the remaining correct matches by the number of workers first. If the numbers of workers are same, GT-hgr determines the order of matches by comparing their aggregate distances. In addition, it iteratively selects one “best” correct match in each iteration until no more correct matches exist.

4. ALGORITHMS IN WORKER-SELECTED-TASKS MODE

Different from server-assigned-tasks mode, in the worker-selected-tasks mode, the servers do not trace the locations of workers and just recommend a task plan for each worker when he/she is querying the available tasks, which indicates a route for the worker to go and conduct tasks by the way. The objective of worker-selected-tasks mode algorithms [10, 11] is to generate a task plan for each worker such that he/she can finish as many tasks as possible on the recommended route, which is called maximum task scheduling (MTS) problem [10]. One example of MTS problem is shown in Figure 3 [10], where worker is located at (6, 5) and 5 tasks A to E are located at five different locations with their deadlines. The result for this example is that the worker can finish at most four tasks following the order $A \rightarrow E \rightarrow C \rightarrow D$.

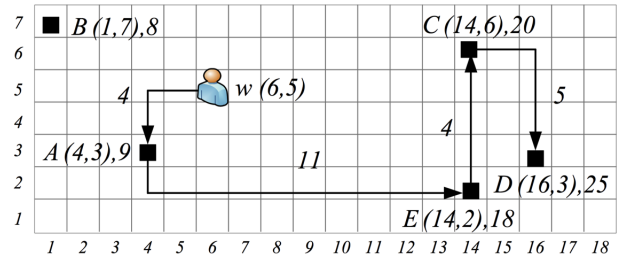


Figure 3: Running example of MTS.

Same with the situation of algorithms in SAT mode, the servers do not have the information of future timestamps and can just recommend routes based on the current information of tasks. As a result, existing algorithms all try to maximize the number of scheduled tasks on the recommended route for each querying worker. In

other words, existing works [10, 11] also solve the problem in a greedy style for each “current” timestamp leading to local optimal results of the entire time period. For a given timestamp, there are two kinds of algorithms that can solve the spatial crowdsourcing problem in WST mode: exact algorithms and heuristic algorithms.

The MTS problem can be reduced from a specialized version of Traveling Salesman Problem (TSP) called sTSP, which is a NP-C problem [10]. As the decision version of MTS is NP-C, the general MTS problem is NP-hard. Exact algorithms, *dynamic programming algorithm* and *branch-and-bound algorithm*, can solve the MTS problem in each timestamp exactly, however, for the entire time period, they still achieve only approximated results. In addition, to improve the efficiency, some heuristic algorithms and progressive algorithms are proposed [10, 11]. In the rest of this section, we will briefly introduce and discuss them.

4.1 Exact Algorithms

As the server in WST mode has almost no control on workers, existing works on spatial crowdsourcing in worker-selected-tasks mode only study how to recommend a longest tasks sequence for each worker such that they can conduct as many tasks as possible, also known as MTS problem. Although MTS problem is proved NP-hard, we still can use dynamic programming algorithm and branch-and-bound algorithm to solve small scale problems.

4.1.1 Dynamic Programming Algorithm for MTS

The first exact algorithm is dynamic programming algorithm (DP) [10], which iteratively expands the sets of tasks in ascending order of set size, and ignores the order of task sequence but examines the sets of tasks. Given a worker w , and a set of tasks Q , let $opt(Q, j)$ be the maximum number of tasks that worker w can complete under the constraints of tasks and starts from his/her current location of w and ends at the task t_j , and R be the corresponding task sequence to achieve the optimum value. In addition, they denote the second-to-last task in R as task t_i . Then, the recurrent formula is given as below

$$opt(Q, t_j) = \begin{cases} 1, & \text{if } |Q| = 1 \\ \max_{t_i \in Q, t_i \neq t_j} \{opt(Q - \{t_j\}, t_i) + \delta_{ij}\}, & \text{otherwise} \end{cases} \quad (4)$$

$$\delta_{ij} = \begin{cases} 1, & \text{if } t_i \text{ can be finished after connecting } t_j \text{ in the end of } R' \\ 0, & \text{otherwise} \end{cases}$$

With the recurrent formula in Equation 4, the algorithm can be implemented based on existing dynamic programming framework.

To further reduce the running time of the dynamic programming algorithm, the Apriori principle [2] can be utilized to remove the invalid sets such that the problem space can be smaller. The observation is that if a task set is invalid, then all of its supersets must be invalid. When exploring the task sets, if one invalid task set is founded, all its supersets can be safely removed. However, when most of the task sets are valid, the optimization strategy may not be effective as the cost of generating candidate sets may surpass the benefits from removing invalid task sets.

4.1.2 Branch-and-Bound Algorithm for MTS

The other exact algorithm for MTS is the branch-and-bound algorithm (BB) [10], which searches the whole problem space with pruning and directing. The search space of branch-and-bound algorithm can be represented as a tree, then the algorithm conducts a depth-first search with effective directing and pruning. Specifically, for each node, the algorithm expands it to a set of candidate

task nodes. One observation is that a node’s candidate task set in the search tree is the subset of its parent’s candidate task set, which can improve the speed of expanding nodes. With the candidate task set of node r , the algorithm can estimate the upper bound ub_r of the maximum task sequence along the node r in the search tree in the equation below

$$ub_r = level(r) + |cand_r| \quad (5)$$

where $level(r)$ indicates the level of node r in the search tree, and $|cand_r|$ represents the size of the candidate task set of node r . Then the algorithm can safely prune the branch of node r when its upper bound ub_r is smaller than the best current known solution $curMax$. To determine the best searching order, the algorithm sorts the current searching branches by their upper bounds (ub) or lower bounds (lb), which can be estimated with approximation algorithms in Section 4.2. In addition, if the upper bound of a node r is less than the lower bound of any other node, the node r can be safely pruned.

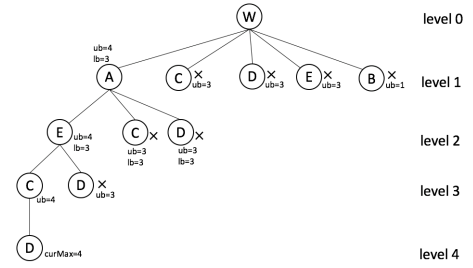


Figure 4: An overview of branch-and-bound algorithm.

Figure 4 displays an overview of the branch-and-bound algorithm of solving the example shown in Figure 3. On level 1, the five nodes are ordered by their upper bounds and node B can be pruned as its upper bound is less than the lower bound of node A. Then after visiting node D on level 4, the algorithm finds the current best known result $curMax = 4$. With $curMax = 4$, the algorithm prunes all other nodes as their upper bounds are all less than 4.

4.2 Heuristic Algorithms

Exact algorithms just can give exact results for each time instance but not the entire time period, and their time complexities and memory consumptions increase exponentially as the number of tasks grows such that they are not efficient enough for real-world applications. In this section, we will briefly introduce three heuristics to return results to the workers quickly [10].

Least expiration time heuristic (LEH). The LEH constructs a task sequence by greedily appending the task with least expiration time to the end of current task sequence. It first orders the tasks by their expiration times, then check each task on the ascending order of their expiration times. If one task can be conducted by the worker, which means the worker can arrive at the location of the task before its deadline, then the algorithm adds the task to the end of the current task sequence. Finally, the task sequence is sent to workers to conduct one by one.

Nearest neighbor heuristic (NNH). The NNH utilizes the spatial proximity between tasks through keeping selecting the nearest valid task to the last added task in the current task sequence, where the valid task means the worker can arrive at its location before its deadline. The heuristic greedily adds more tasks to the end to the task sequence until no more tasks can be selected, then it returns the task sequence to the worker who is querying the available tasks.

Most promising heuristic (MPH). The MPH is a heuristic for the branch-and-bound algorithm in Section 4.1.2 to choose the most promising branches when it is exploring the search tree, where the most promising branch for each level can be the branch having the nodes with the highest upper bound at that level. In addition, the MPH just reports the first found candidate task sequence.

As the heuristic algorithms run fast, on real-world application, the system can run the three heuristic algorithms at the same time (HA), and just report the best result to improve the accuracy of the final result but without harming the user experience of workers.

4.3 Progressive Algorithms

The idea of progressive algorithms (PRS) is to report a small number of spatial tasks to a worker quickly at the beginning, and then to keep incrementally building the rest of the task sequence off-line and report the newly added tasks to the worker before he/she finishes all the tasks already reported to him/her. Under this framework, one progressive algorithm can use approximation algorithms to response one worker very fast at the beginning, then utilizes one exact algorithm to progressively construct the rest part of the task sequence.

The advantage of progressive algorithms is that they can response a worker faster than exact algorithms and report more accurate results than heuristic ones. On the other hand, the potential tasks for a worker may be promoted to other workers when he/she is conducting the initial tasks, and he/she cannot see the entire task sequence at the beginning which may lead to a worse user experience compared to that of the other WST algorithms.

5. EXPERIMENTAL STUDY

Table 2: Experiments Settings.

Parameters	Values
number of tasks m	7.5K , 10K, 12.5K, 15K, 17.5K
number of workers n	7.5K , 10K, 12.5K, 15K, 17.5K
expiration time range $[rt^-, rt^+]$	[1, 2] , [2, 3], [3, 4], [4, 5]
required answers range $[a^-, a^+]$	[1, 3], [3, 5] , [5, 7], [7, 9]
required confidences range $[q^-, q^+]$	[0.65, 0.7], [0.75, 0.8] , [0.8, 0.85], [0.85, 0.9]
reliability range $[r^-, r^+]$	[0.65, 0.7], [0.75, 0.8] , [0.8, 0.85], [0.85, 0.9]
capacity range $[c^-, c^+]$	[2, 3] , [3, 4], [4, 5], [5, 6]
length of side range $[s^-, s^+]$	[0.05, 0.1] , [0.1, 0.15], [0.15, 0.2], [0.2, 0.25]

5.1 Experiments Setup

Data Sets. We use both real and synthetic data to test task assignment methods on Server-Assigned-Task mode (SAT) and Worker-Select-Task mode (WST).

Specifically, for real data, we utilize historical data from two sources: Gowalla [8] and Foursquare [24]. Gowalla is a location-based social network, which contains check-in records of users, as well as their locations and timestamps (within a period of 5 months, from Feb. 2009 to Oct., 2010). Foursquare is a local search and discovery service mobile app which provides search results for its users, and its dataset includes long-term (about 18 months from April 2012 to September 2013) global-scale check-in data. Specifically, we extract the records in the area of Los Angeles (with latitude from 33.692965° to 34.353218° and longitude from -118.661469° to -118.161934°). Then we treat Gowalla data records as check-in data of workers (locations and time) in the spatial crowdsourcing, meanwhile we utilize Foursquare location records as the required locations of spatial tasks. We use the data records in one day as one time instance and calibrate the timestamps of two datasets to match each other. In addition, as the number of Foursquare daily data records is much larger than that of

Gowalla dataset, to match the two data sources, we uniformly sample 300 records from daily records of Gowalla dataset and add back to the data set. For each real dataset, we generate 20 time instances.

For synthetic data, we generate locations of workers and tasks in a 2D data space $[0, 1]^2$ following Uniform (UNIF), Gaussian (GAUS), Skewed (SKEW). For Uniform distribution, we uniformly generate the locations of tasks/workers in the 2D data space. For Gaussian distribution, we generate the locations of tasks/workers in a Gaussian cluster (centered at (0.5, 0.5) with variance = 0.2^2). Similarly, we also generate tasks/workers with the Skewed distribution locating 90% of them into a Gaussian cluster (centered at (0.5, 0.5) with variance = 0.2^2), and distribute the rest workers/tasks uniformly. To simulate the synthetic data and the other properties of real data, we use a toolbox, SCAWG [22], to generate data records for each time instance. In total, for each synthetic dataset, we generate 50 time instances.

For both real and synthetic data sets, we simulate the working ranges each worker as squares whose centers are at the locations of workers, and the length of sides of the squares are generated with Gaussian distribution within range $[s^-, s^+]$ [15, 23], for $s^-, s^+ \in (0, 1)$. For the count of required answers of each task and the capacity of each worker, we generate them following the Gaussian distributions within the range $[a^-, a^+]$ and the range $[c^-, c^+]$ respectively [15, 23]. Meanwhile, for the required confidence of each task and the reliability of each worker, we produce them following the Gaussian distributions within the range $[q^-, q^+]$ and the range $[r^-, r^+]$ respectively [16]. For temporal constraints of tasks, we also generate the deadlines of tasks, e , within range $[rt^-, rt^+]$ with Gaussian distribution [6, 7, 10]. Here, for Gaussian distributions, we linearly map data samples within $[-1, 1]$ of a Gaussian distribution $\mathcal{N}(0, 0.2^2)$ to the target ranges.

Approaches and Measures. Table 2 depicts our experimental settings, where the default values of parameters are in bold font. In each set of experiments, we vary one parameter, while setting other parameters to their default values. For each experiment, we report the running time, the number of assigned workers, the number of finished tasks (the number of assigned workers is not less than the number of required workers of the task), the number of *confident finished tasks* (confident finished task means its aggregation reputation score calculated by Equation 3 is not less than its required confidence level), and average worker moving distance of tested approaches, which includes the algorithms for SAT mode: maximum flow based greedy algorithm (G-greedy), maximum flow with least location entropy priority heuristic algorithm (G-llep), maximum flow with nearest neighbor priority heuristic algorithm (G-nnp), greedy algorithm for trustworthy query (GT-greedy), heuristic-enhanced greedy algorithm for trustworthy query (GT-hgr), sampling-based algorithm (RDB-sam) and divide-and-conquer-based algorithm (RDB-d&c), and the algorithms for WST mode: dynamic programming algorithm (DP), branch-and-bound algorithm (BB), heuristic algorithm for WST (HA, here we run three heuristic algorithms introduces in Section 4.2 and report the best result of the results of them) and progress algorithm (PRS). Table 1 summarizes all the tested algorithms, where E is the number of valid worker-and-task pairs, $\max|f|$ is the size of the maximum flow, m is the number of tasks and n is the number of workers. All our experiments were run on an Intel Xeon X5675 CPU @3.07 GHZ with 32 GB RAM in Python.

5.2 Experimental Results

5.2.1 Experimental Results on Synthetic Data

—×— GT-greedy —△— GT-hgr —○— G-greedy —*— G-llep —+— G-nnp —□— RDB-d&c —◇— RDB-sam

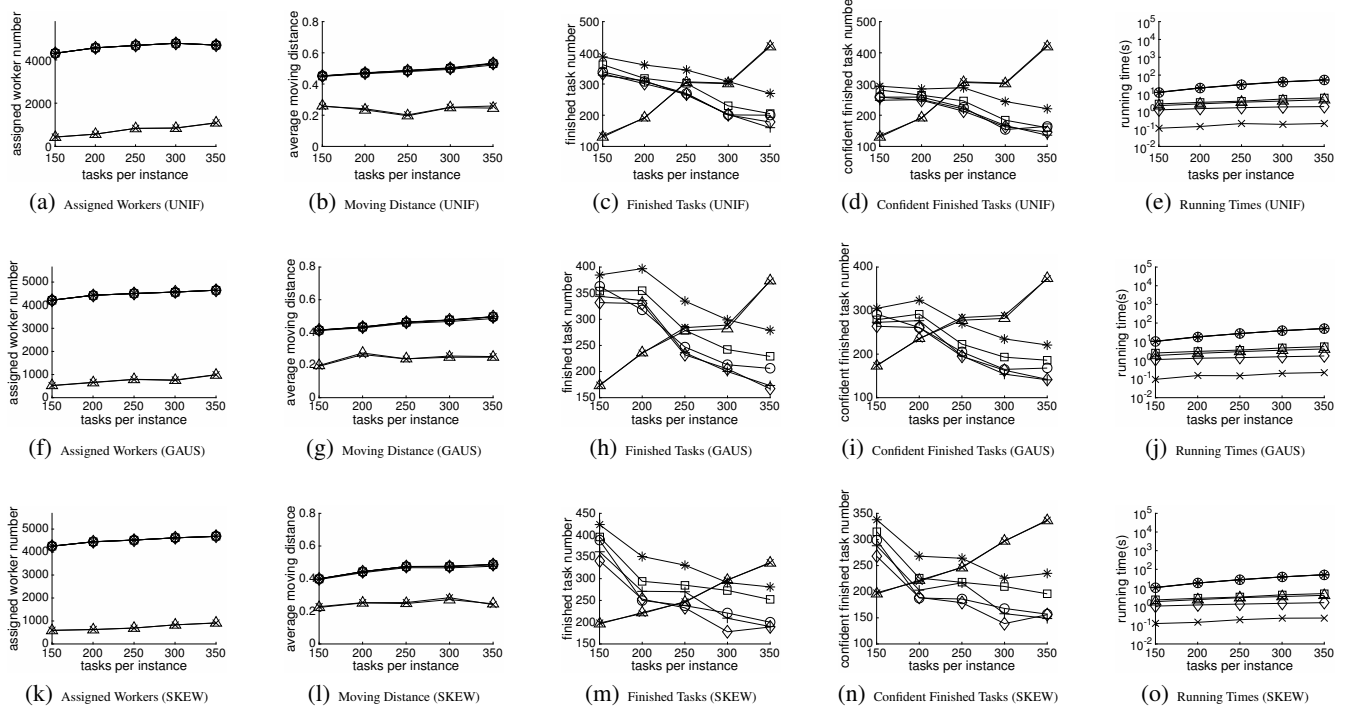


Figure 5: Effects of Task Number Per Round m (SAT Mode, Synthetic).

Table 1: Algorithms Comparison

Algorithms	Time Complexity	Assignment Mode	Objective Goal	Randomization
G-greedy [15]	$O(E \max f)$	SAT	maximize the number of assigned tasks	Deterministic
G-llep [15]	$O(E \max f)$	SAT	maximize the number of assigned tasks	Heuristic
G-nnp [15]	$O(E \max f)$	SAT	maximize the number of assigned tasks	Heuristic
GT-greedy [16]	-	SAT	maximize the number of correct matches	Randomized
GT-HGR [16]	-	SAT	maximize the number of correct matches	Heuristic
RDB-d&c [7]	$O(m \cdot n^2)$	SAT	maximize the global diversity and reliability	Heuristic
RDB-sam [7]	-	SAT	maximize the global diversity and reliability	Randomized
DP [10]	$O(n \cdot m^2 \cdot 2^m)$	WST	maximize the number of performed tasks	Deterministic
BB[10]	$O(n \cdot m!)$	WST	maximize the number of performed tasks	Deterministic
HA [10]	$O(n \cdot \log(m))$	WST	maximize the number of performed tasks	Heuristic
PRS [10]	-	WST	maximize the number of performed tasks	Heuristic

In this subsection, we show the performances of tested approaches in both WST mode and SAT mode on synthetic dataset by varying the number of tasks per time instance m , the number of workers per time instance n and the range of task durations rt .

Effect of the Number of Tasks m . Figure 5 and Figure 6 show the effect of the number m of spatial tasks on the performances of tested approaches, where we vary m from 1K to 10K. For each figure, we show the results when the locations of workers/tasks both follow Uniform (UNIF) distribution, Gaussian (GAUS) distribution and Skewed (SKEW) distribution respectively.

In the first column of Figure 5, all the tested SAT approaches can assign more workers when the number of tasks per time instance increases, as there are more suitable tasks for each worker to select. For the GT algorithms (GT-greedy and GT-HGR), as they only assign correct matches, where a correct match is a pair of a task and a set of workers that can achieve an aggregate reputation score no less than the confidence level of the task, they assign much

fewer workers compared with other approaches that do not concern the reliability of each individual task. Although RDB algorithms (RDB-sam and RDB-d&c) can maximize the minimum reliability of all the tasks, the guarantee is too weak compared with the algorithms for the trustworthy queries. Thus, RDB algorithms can assign much more workers than GT algorithms (GT-greedy and GT-HGR). Similarly, all the tested WST algorithms can also assign more workers when m increases, as shown in the first column of Figure 6.

In the second column of Figure 5, the assigned workers of all the tested approaches will have higher average moving distances for larger m . The reason is that the approaches select tasks in perspectives different from proximity of tasks. When the number of tasks per time instance increases, they may select most suitable tasks located further. Moreover, we find G-nnp algorithm just can reduce very limited average moving distances of the worker-and-task pairs, which means the nearest neighbor priority heuristic is not quite ef-

—x— PRS —△— DP —○— BB —*— HA

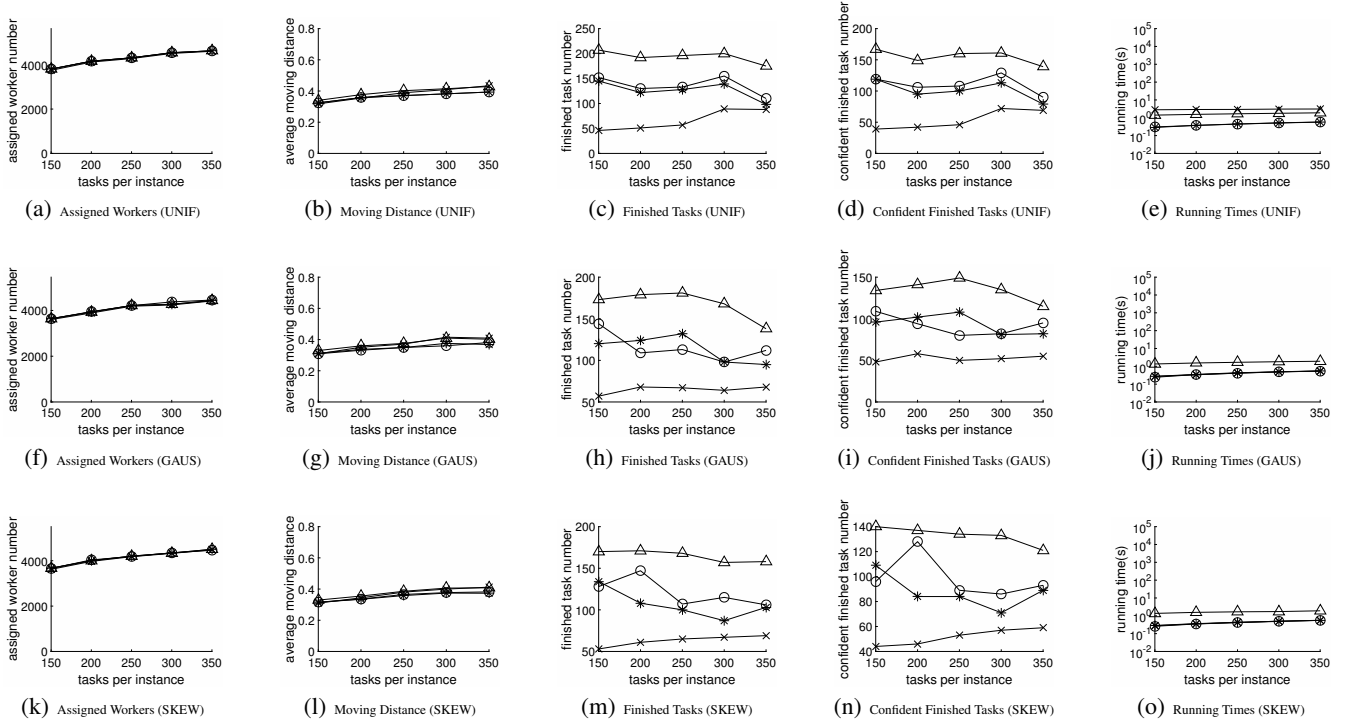


Figure 6: Effects of Task Number Per Round m (WST Mode, Synthetic).

fective on synthetic data. In addition, as the GT algorithms assign much fewer workers, the average moving distance of the results achieved by them is small. In the second column of Figure 6, the average moving distance of the results of WST mode algorithms also increases when m gets larger and the differences in average moving distance of WST mode algorithms are very small.

In the third column of Figure 5, the numbers of finished tasks achieved by G algorithms and RDB algorithms decrease with the increase of the number of tasks per time instance. The reason is that G algorithms and RDB algorithms both do not concern the minimum required number of answers by each task. When the number of tasks increases, the workers are distributed to more tasks and the average workers for each task will decrease, which leads to the number of finished tasks decreases. However, for GT algorithms, they just assign correct matches, which can guarantee each assigned task will have a set of workers to satisfy the minimum number of workers it requires. Meanwhile, when the number of tasks increases, GT algorithms will produce more correct matches as more suitable tasks are available to be selected such that the number of finished tasks of GT algorithms increases. The similar results can be observed in the forth column of Figure 5 due to the same reason. As for the results of finished tasks of WST algorithms shown in the third column of Figure 6, DP algorithm can complete most tasks and PRS algorithm can complete least tasks, as DP can solve time instance problem exactly. When m increases, PRS algorithm will complete more tasks, however, other WST algorithms will complete fewer tasks. Similar results can be achieved when the confident of tasks is concerned, as shown in the forth column of Figure 5 and Figure 6. In addition, we notice that not all the finished tasks' confidence values are satisfied, as the aggregation reputation scores may be small even the required numbers of workers of finished tasks are satisfied.

In the last column of Figure 5, the running time of all the tested SAT approaches increases with the increase of the number of the tasks per time instance. The running time of RDB-d&c increases as it needs to divide the original problem based on the locations of tasks and more tasks means more divide and conquer levels. RDB-d&c algorithms needs more time to calculate results than RDB-sam, but still much faster than G algorithms. The speeds of G algorithms are similar with each other. In the last column of Figure 6, DP algorithm runs much slower than other WST algorithms. The speeds of other WST algorithms are similar.

To compare the algorithms in SAT mode and WST mode together, we select three algorithms performing well from each category and place the results of them in the same figures to compare clearly. Specifically, we select GT-hgr, G-llep and RDB-d&c from algorithms in SAT mode, and select DP, HA and PRS from algorithms in WST mode. In the following discussion, we just show the results of the six selected algorithms. In addition, we just show the results on the data set that the locations of workers and tasks follow the SKEW distribution. For more detail results of all the tested algorithms with other distribution data sets, please refer to our technical report [5].

Effect of the Number of Workers n . Figure 7 shows the effect of the number n of spatial workers on the performances of tested approaches, where we vary n from 1K to 10K.

In Figure 7(a), the number of assigned workers increases for all the approaches with the increase of the number of workers per time instance. The more workers exist, the more workers can be assigned. RDB-d&c and G-llep can assign more workers than other algorithms. Moreover, GT-hgr algorithm assigns much fewer workers than other algorithms as GT-llep algorithm just select correct matches. WST algorithms assign similar number of workers, as they are worker-oriented.

—× GT-hgr —△ G-llep —○ RDB-d&c —* PRS —+ DP —□ HA

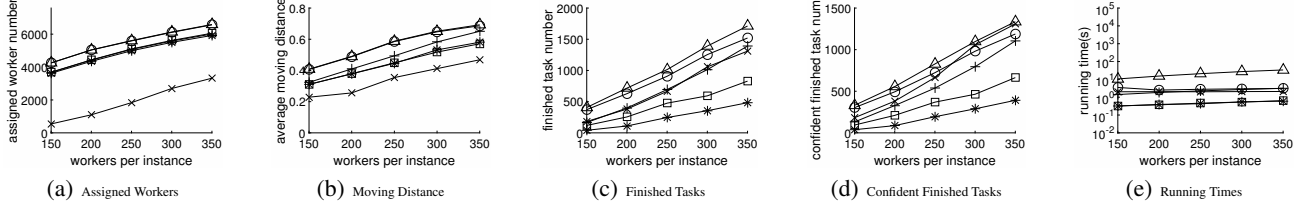


Figure 7: Effects of Number of Workers Per Round n (SKEW).

—× GT-hgr —△ G-llep —○ RDB-d&c —* PRS —+ DP —□ HA

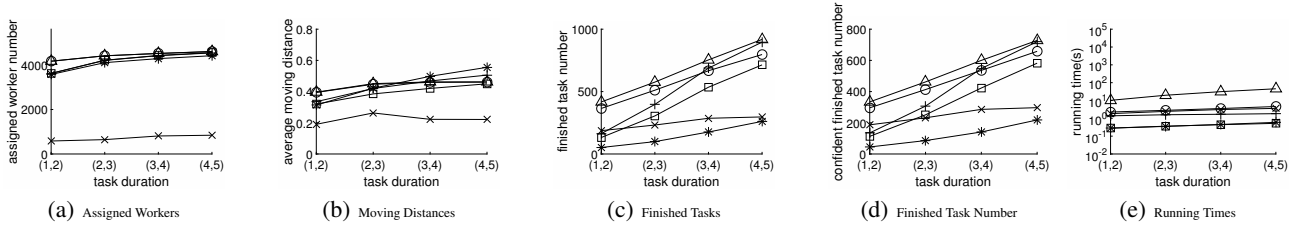


Figure 8: Effects of Tasks' Expiration Times rt_j (SKEW).

—× GT-hgr —△ G-llep —○ RDB-d&c —* PRS —+ DP —□ HA

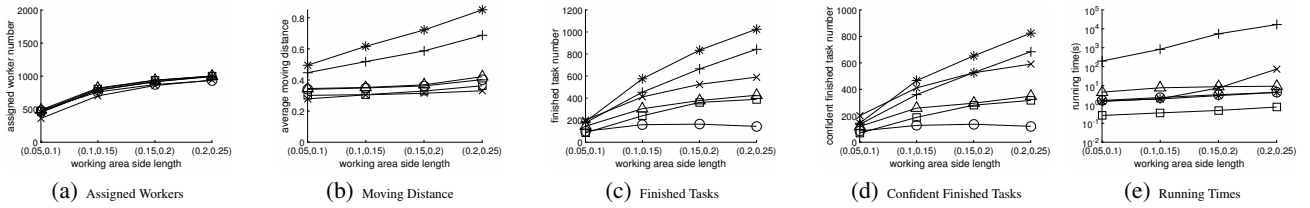


Figure 9: Effects of Workers' Working Ranges s_i (Real).

As shown in Figure 7(c), when m increases, all the tested approaches can complete more tasks, as more worker labor is available. WST algorithms can complete fewer tasks than the SAT algorithms. G-llep algorithms can complete more tasks than other algorithms. The reason is when more workers exist, the location entropy will be more accurate and the heuristic can make use of the pattern that workers regularly appear in some areas. In Figure 7(d), all the approaches can complete more tasks in terms of confidence of tasks when there are more workers available. Moreover, the increase rate of (confident) finished tasks of GT-hgr increases faster than other approaches. As the GT-hgr algorithms only assign correct matches, when more workers are available, the number of correct matches will increase exponentially leading to the number of (confident) finished tasks with confidence increases similarly.

In Figure 7(e), only the running time of G-llep algorithms increases dramatically when the number of workers per time instance increases, as the complexity of maximum flow algorithm increases linearly with respect to the number of edges of the graph, which increases superlinearly w.r.t n . PRS and HA are faster than other algorithms, as PRS can quickly assign initial tasks at the beginning and HA just assigns tasks based on very simple heuristics (e.g., selecting next nearest neighbor). G-llep is slower than other five algorithms.

In the rest of this section, we will just discuss interesting results, due to the space limitations. For detailed description of all the results, please refer to Appendix B of our technical report [5].

Effect of the Range, $[rt^-, rt^+]$, of Tasks' Expiration Times rt_j .

In Figure 8(a), when the range of tasks' expiration times rt_j increases, the number of assigned workers of the tested approaches also increases. The reason is that each task will last more time in-

stances such that the number of tasks in the later time instances increases leading to that workers have more valid tasks to choose or assign. GT-hgr assigns much fewer workers than other tested algorithms does, as it only assigns correct matches.

In Figure 8(b), the average moving distance of the results of WST algorithms increases slightly when rt_j increases. The reason is when the range of task durations increases, workers can arrive at tasks located at farther locations leading to WST algorithms schedule workers to farther tasks.

As shown in Figure 8(c), when the range of rt_j increases, all the tested approaches can complete more tasks, as each task can be reached by more workers before its deadline. GT-hgr still can complete fewer tasks than other algorithms except PRS, as GT-hgr algorithms just assign correct matches. Moreover, G-llep algorithm can complete more tasks than other algorithms. In addition, RDB-d&c algorithms can also complete many tasks but slightly less than G-llep algorithm. Similarly, when we consider the confidence of tasks as shown in Figure 8(d), the G-llep algorithm still can complete the most tasks than other algorithms.

5.2.2 Experiments on Real Data

In this subsection, we show the results on the real data set and vary the range of workers' working ranges s_i , the range of tasks' required confidences q_j and the range of workers' reliabilities r_i . Due to space limitations, please refer to experimental results with respect to other parameters (e.g., $[c^-, c^+]$ and $[a^-, a^+]$) in Appendix A of our technical report [5].

Effect of the Range, $[s^-, s^+]$, of Workers' Working Ranges s_i .

When the range of workers' working areas increases, there will be more available tasks located in the working area of each worker,

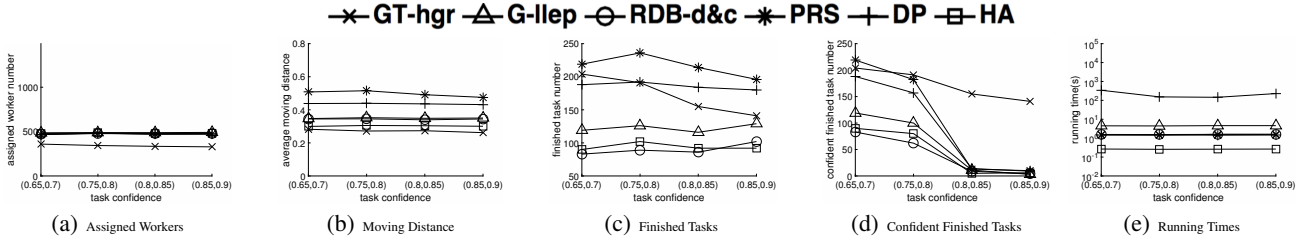


Figure 10: Effects of Tasks' Required Confidences q_j (Real).

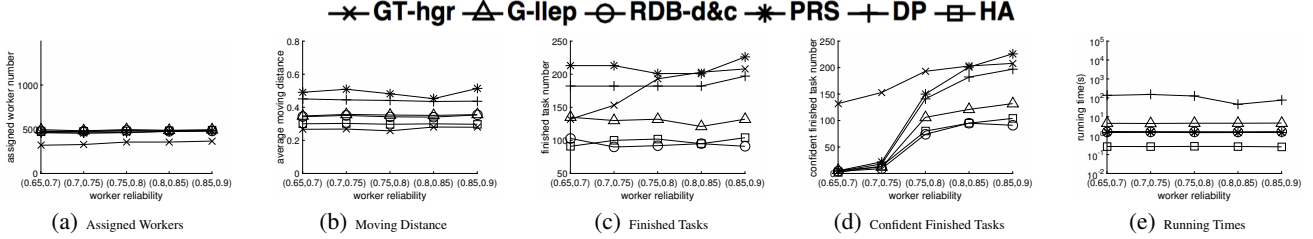


Figure 11: Effects of Workers' Reliabilities r_i (Real).

which leads to the number of valid worker-and-task pairs increases.

In Figure 9(a), when the range of s_i increases, all the tested approaches can assign more workers, as each worker can reach more tasks leading to more valid worker-and-task pairs and can be assigned to suitable tasks with a higher probability.

In Figure 9(b), as the range of s_i increases, the average moving distances of the results achieved by all the tested approaches increase obviously. The reason is when the working area of each worker becomes larger, the worker can reach tasks located further.

In Figure 9(c), all the tested approaches can complete more tasks when the range of s_i increases, as the number of the valid worker-and-task pairs increases. Unlike the experimental results on the synthetic data, PRS can complete more tasks than other algorithms on real data sets. Meanwhile, RDB-d&c can complete fewer tasks than other algorithms. The reason is the skewness of real datasets is higher than that of synthetic datasets, which leads to that WST algorithms can more easily plan tasks for each worker.

In Figure 9(e), the running time of all the tested approaches increases when the range of s_i increases. DP algorithm needs much more time than other approaches as for each worker the computation complexity is $O(m^2 \cdot 2^m)$. When the skewness is high, there will be many reachable tasks (large m). As PRS can quickly assign initial tasks to workers at the beginning, it runs fast and is just slower than HA.

Effect of the Range, $[q^-, q^+]$, of Tasks' Required Confidences q_j . When the range of tasks' required confidences changes, only GT-hgr will be affected by all the measures and other algorithms will only be affected in the number of confident finished tasks.

In Figure 10(c), GT-llep algorithms can assign fewer workers when the range of q_j gets larger. When the required confidence of task gets higher, the number of correct matches will decrease leading to the number of assigned workers decreases.

In Figure 10(d), when the range of q_j increases, all the tested approaches can achieve fewer finished tasks w.r.t. the confidences of tasks. We notice that although G-llep and RDB-d&c can complete more tasks than GT-hgr does when q_j is small, GT-hgr can complete more tasks when q_j becomes large (e.g., 0.85 to 0.9), which shows the effectiveness of the trustworthy query.

Effect of the Range, $[r^-, r^+]$, of Workers' Reliabilities r_i . In Figure 11(c), GT-hgr can assign fewer workers when the range of r_i gets larger. When the reliabilities of workers get higher, the

number of correct matches will increase leading to the number of assigned workers increases.

In Figure 11(d), when the range of r_i increases, all the tested approaches can achieve results with more finished tasks w.r.t. the confidences of tasks. We notice that although G-llep and RDB-d&c can complete more tasks than GT-hgr when r_i is large, GT-hgr can complete more tasks when r_i is small (e.g., 0.65 to 0.7), which shows the effectiveness of the trustworthy query. When the worker labor is scarce, GT-hgr can guarantee the correctness of the assigned tasks with respect to the confidence of them.

5.3 Summary

- Although WST algorithms have no global pictures and only greedily schedule tasks plan for each requesting worker, they can complete more tasks than SAT algorithms on real datasets. The difference between real dataset and synthetic dataset is the tasks in real dataset are crowded in the positions of workers such that each worker can easily complete a large number of tasks before their deadlines. Under that situation, WST algorithms can complete more tasks. In addition, among WST algorithms, PRS is quite effective and efficient on real dataset.
- GT-hgr can guarantee the confidence of tasks, especially when the required confidences of tasks q_j are high and/or the reliabilities of workers r_i are low. When confidences q_j are low and/or reliabilities r_i are high, there is no need to particularly care the confidences of tasks and the tasks can be finished better with other algorithms, such as PRS.
- G-nnp heuristic is not that efficient. The average moving distances of the results achieved by G-nnp are just a little bit smaller than other algorithms that can complete close number of tasks. As a result, for the results on the entire time period, G-nnp algorithm cannot outperform other algorithm observably.
- RDB-g&c works well on synthetic data. Particularly, it can complete more tasks than other algorithms except G-llep, however, it runs much faster than G-llep.
- Any task hoping to complete more tasks will lead more moving distances for each worker. The reason is that tasks close

to workers are easily completed by all the algorithms, then an algorithm can complete more tasks only when it can assign workers to tasks located far from the locations of workers.

6. CONCLUSION

In this paper, we present a comprehensive experimental comparison of most existing algorithms on task assignment in spatial crowdsourcing. Specifically, we first give some general definitions about spatial workers and spatial tasks based on definitions in the existing works studying task assignment problems in spatial crowdsourcing such that the existing algorithms can be applied on same synthetic and real data sets. We uniformly implement tested algorithms in both SAT and WST modes. With the experimental results of the tested algorithms on synthetic and real datasets, we show the effectiveness and efficiencies of the algorithms through their performances on five important metrics. We find that PRS algorithm is in fact effective and efficient on real dataset, although WST algorithms have no global pictures of the whole system. Currently, the existing works on WST mode are much fewer than that on SAT mode. With the findings of our paper, future research on this area may pay more attention on WST mode algorithms and practical implementations also can concern more about WST mode algorithms.

7. REFERENCES

- [1] <http://www.gmissionhkust.com>.
- [2] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.
- [3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows. Technical report, DTIC Document, 1988.
- [4] Z. Chen, R. Fu, Z. Zhao, Z. Liu, L. Xia, L. Chen, P. Cheng, C. C. Cao, and Y. Tong. gmission: A general spatial crowdsourcing platform. *Proceedings of the VLDB Endowment*, 7(13), 2014.
- [5] P. Cheng, X. Jian, and L. Chen. Task assignment on spatial crowdsourcing (technical report). <http://arxiv.org/abs/1605.09675>.
- [6] P. Cheng, X. Lian, L. Chen, J. Han, and J. Zhao. Task assignment on multi-skill oriented spatial crowdsourcing. *Knowledge and Data Engineering, IEEE Transactions on*, 2015.
- [7] P. Cheng, X. Lian, Z. Chen, R. Fu, L. Chen, J. Han, and J. Zhao. Reliable diversity-based spatial crowdsourcing by moving workers. *Proceedings of the VLDB Endowment*, 8(10):1022–1033, 2015.
- [8] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1082–1090. ACM, 2011.
- [9] J. Cranshaw, E. Toch, J. Hong, A. Kittur, and N. Sadeh. Bridging the gap between physical location and online social networks. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*, pages 119–128. ACM, 2010.
- [10] D. Deng, C. Shahabi, and U. Demiryurek. Maximizing the number of worker’s self-selected tasks in spatial crowdsourcing. In *Proceedings of the 21st SIGSPATIAL GIS*, pages 314–323, 2013.
- [11] D. Deng, C. Shahabi, U. Demiryurek, and L. Zhu. Task selection in spatial crowdsourcing from worker’s perspective. *GeoInformatica*, 20:529–568, 2016.
- [12] U. U. Hassan and E. Curry. A multi-armed bandit approach to online spatial task assignment. In *Ubiquitous Intelligence and Computing, 2014 IEEE 11th Intl Conf on and IEEE 11th Intl Conf on and Autonomic and Trusted Computing, and IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UTC-ATC-ScalCom)*, pages 212–219. IEEE, 2014.
- [13] H. Hu, Y. Zheng, Z. Bao, G. Li, J. Feng, and R. Cheng. Crowdsourced poi labelling: Location-aware result inference and task assignment. *ICDE*, 2016.
- [14] L. Kazemi and C. Shahabi. A privacy-aware framework for participatory sensing. *SIGKDD Explorations Newsletter*, 13(1):43–51, 2011.
- [15] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *Proceedings of the 21st SIGSPATIAL GIS*, pages 189–198, 2012.
- [16] L. Kazemi, C. Shahabi, and L. Chen. Geotrucrowd: trustworthy query answering with spatial crowdsourcing. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 314–323. ACM, 2013.
- [17] S. H. Kim, Y. Lu, G. Constantinou, C. Shahabi, G. Wang, and R. Zimmermann. Mediaq: mobile multimedia management system. In *Proceedings of the 5th ACM Multimedia Systems Conference*, pages 224–235. ACM, 2014.
- [18] J. Kleinberg and É. Tardos. *Algorithm design*. Pearson Education India, 2006.
- [19] Y. Li, M. L. Yiu, and W. Xu. Oriented online route recommendation for spatial crowdsourcing task workers. In *Advances in Spatial and Temporal Databases*, pages 137–156. Springer, 2015.
- [20] R. G. Michael and S. J. David. Computers and intractability: a guide to the theory of np-completeness. *WH Free. Co., San Fr*, 1979.
- [21] L. Pournajaf, L. Xiong, V. Sunderam, and S. Goryczka. Spatial task assignment for crowd sensing with cloaked locations. In *Mobile Data Management (MDM), 2014 IEEE 15th International Conference on*, volume 1, pages 73–82. IEEE, 2014.
- [22] H. To, M. Asghari, D. Deng, and C. Shahabi. Scawg: A toolbox for generating synthetic workload for spatial crowdsourcing. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 1–6. IEEE, 2016.
- [23] H. To, C. Shahabi, and L. Kazemi. A server-assigned spatial crowdsourcing framework. *ACM Transactions on Spatial Algorithms and Systems*, 1(1):2, 2015.
- [24] D. Yang, D. Zhang, L. Chen, and B. Qu. Nationtelescope: Monitoring and visualizing large-scale collective behavior in lbsns. *Journal of Network and Computer Applications*, 55:170–180, 2015.
- [25] M. L. Yiu and N. Mamoulis. Efficient processing of top-k dominating queries on multi-dimensional data. In *Proceedings of the 33rd international conference on Very large data bases*, pages 483–494. VLDB Endowment, 2007.

APPENDIX

—× GT-hgr —△ G-llep —○ RDB-d&c —* PRS —+ DP —□ HA

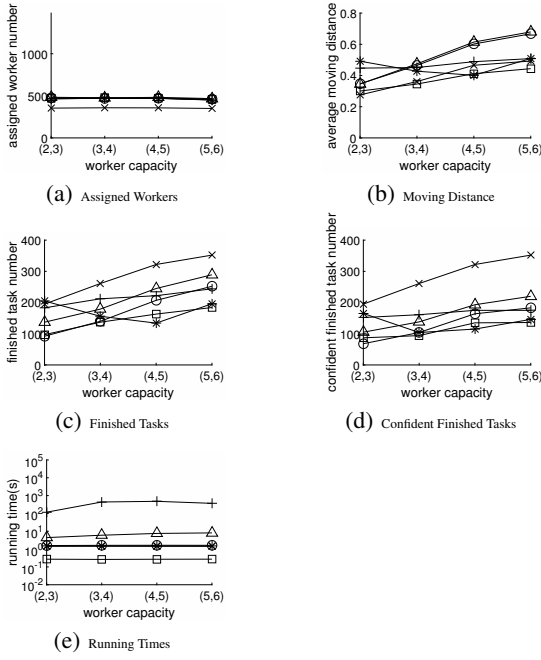


Figure 12: Effects of Workers' Capacities c_i (Real).

—× GT-hgr —△ G-llep —○ RDB-d&c —* PRS —+ DP —□ HA

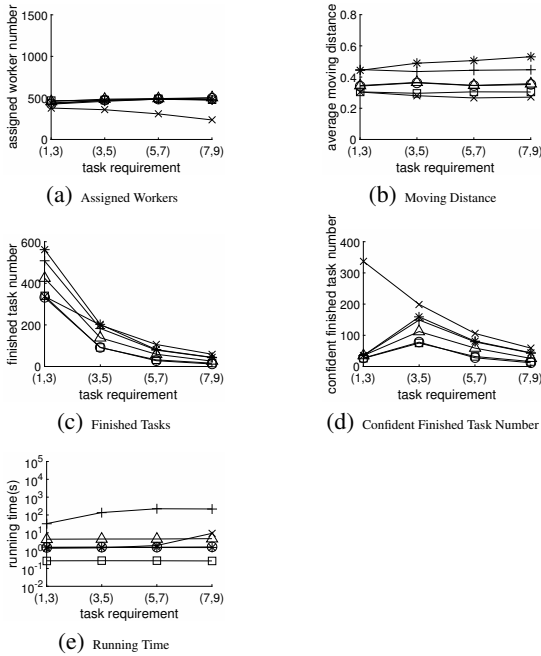


Figure 13: Effects of Tasks' Required Answer Count a_j (Real).

A. Effect of Other Parameters

The Efficiency vs. the Range, $[c^-, c^+]$, of Workers' Capacities c_i . Due to the saturation of the worker labor, when the capacities c_i of workers increase, the number of assigned workers will not increase as shown in Figure 12(a). On the other hand, as each worker can conduct more tasks when the range of c_i increases, each worker need to move more distances to finish more tasks as shown in Figure 12(b). However, we find G-llep in fact sacrifices the efficiency of moving distances to finish more tasks, which means when some

tasks are located in locations with low location entropies, the algorithm will assign these tasks with higher priorities such that the total moving distances of workers may increase.

The numbers of finished tasks of the tested algorithms are shown in Figure 12(c). When the range of c_i increases, the numbers of finished tasks of all the algorithm except PRS increase. The effect of the range of c_i is not stable for PRS. As shown in Figure 12(c) and Figure 12(d), the number of finished tasks of PRS decreases first and increases later when the range of c_i increases from (2, 3) to (5, 6). Moreover, GT-hgr can finish more tasks than other tested approaches.

For the running times of the tested approaches as shown in Figure 12(e), DP needs more time than other approaches.

The Efficiency vs. the Range, $[a^-, a^+]$, of Tasks' Required Answers Count a_j . As show in Figure 13(a), when the range of a_j increases, GT-hgr assigned fewer workers and other approaches assigned similar workers. As the worker labor does not increase, when the range of a_j increases, the moving distances of the workers in all the results of the testes approaches do not change, as shown in Figure 13(b).

For the number of finished tasks as shown in Figure 13(c), when the range of a_j increases, all the approaches can finish fewer tasks as the worker labor does not increase. For the number of confident finished tasks as shown in Figure 13(d), different approaches performed quite different. When the range of a_j increases, the number of confident finished tasks achieved by GT-hgr decreases. The reason is when the range of a_j increases, more correct matches appear, then the greedy-based approach, GT-hgr, has a lower performance. For other approaches, when the range of a_j is too small, like (1, 3), other approaches can rarely finish tasks confidently with just less than 3 workers as they do not care the correctness of the assignment. When the range of a_j increases a little, they can finish more tasks. But when the range of a_j becomes too large, the worker labor is spread to many different tasks, then the number of confident finished tasks decreases.

B. Results on Other Distributions Here we show the results of tested approaches in SAT mode or WST mode with SKEW or GAUS distributions.

—× GT-hgr —△ G-llep —○ RDB-d&c —* PRS —+ DP —□ HA

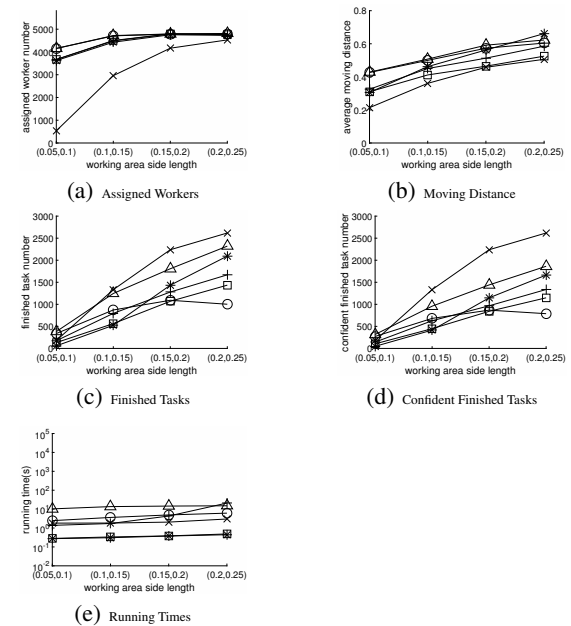


Figure 14: Effects of Workers' Working Ranges s_i (SKEW).

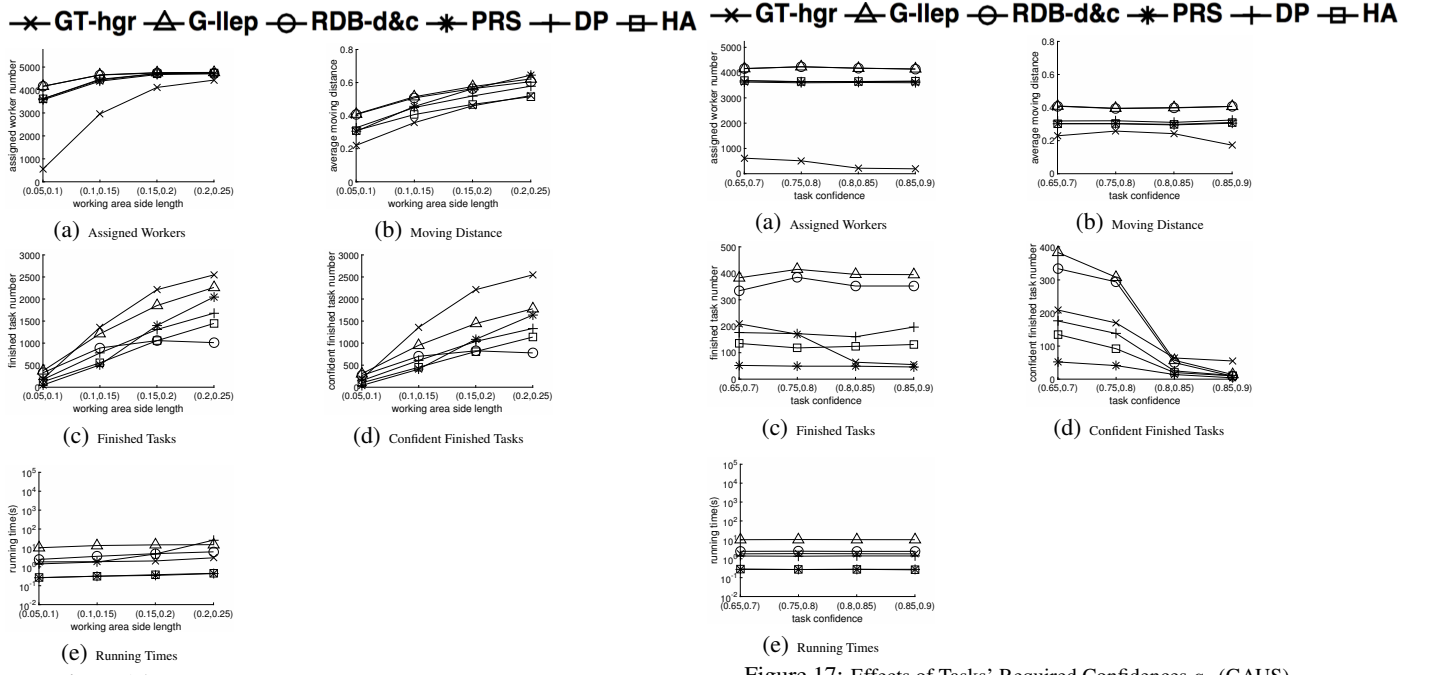


Figure 15: Effects of Workers' Working Ranges s_i (GAUS).

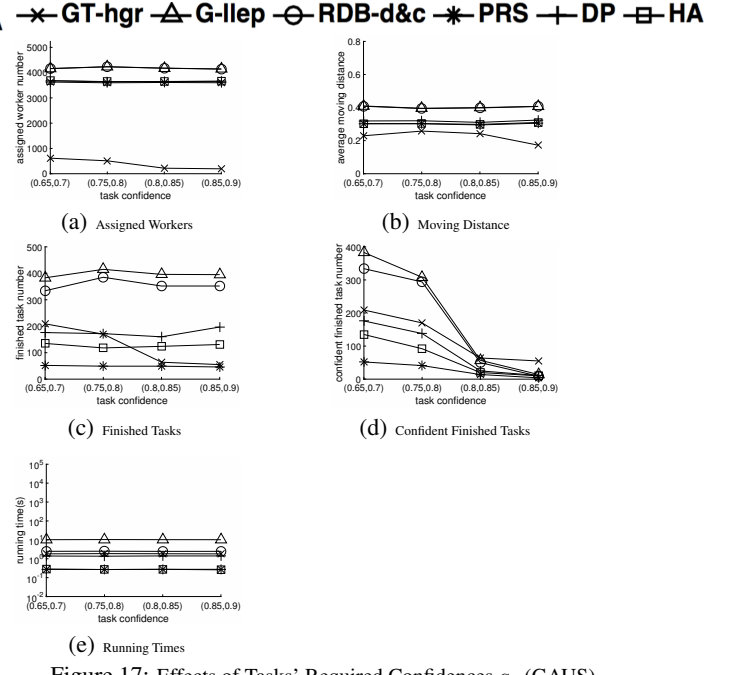


Figure 17: Effects of Tasks' Required Confidences q_j (GAUS).

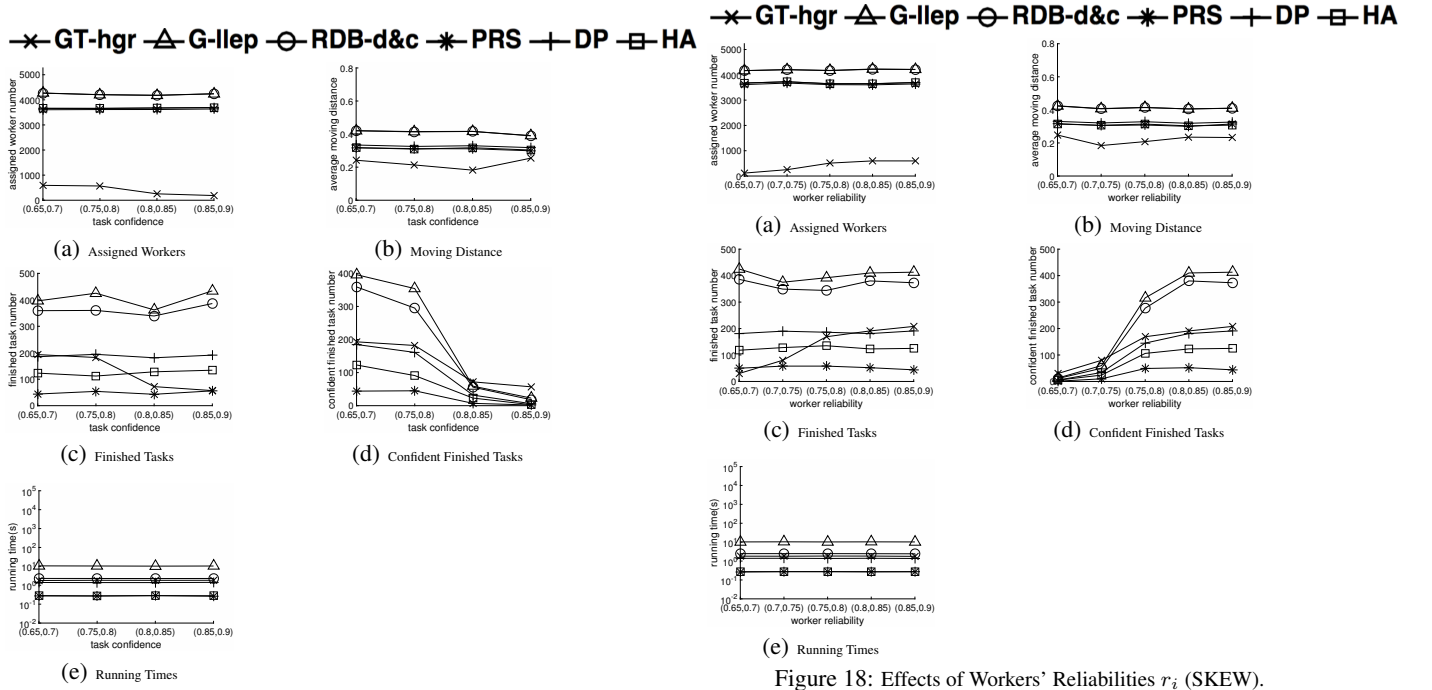


Figure 16: Effects of Tasks' Required Confidences q_j (SKEW).

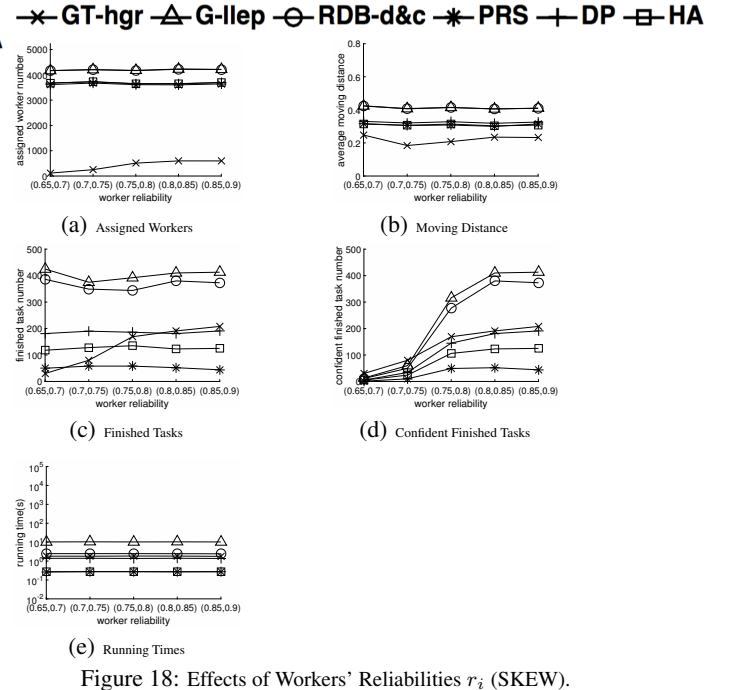


Figure 18: Effects of Workers' Reliabilities r_i (SKEW).

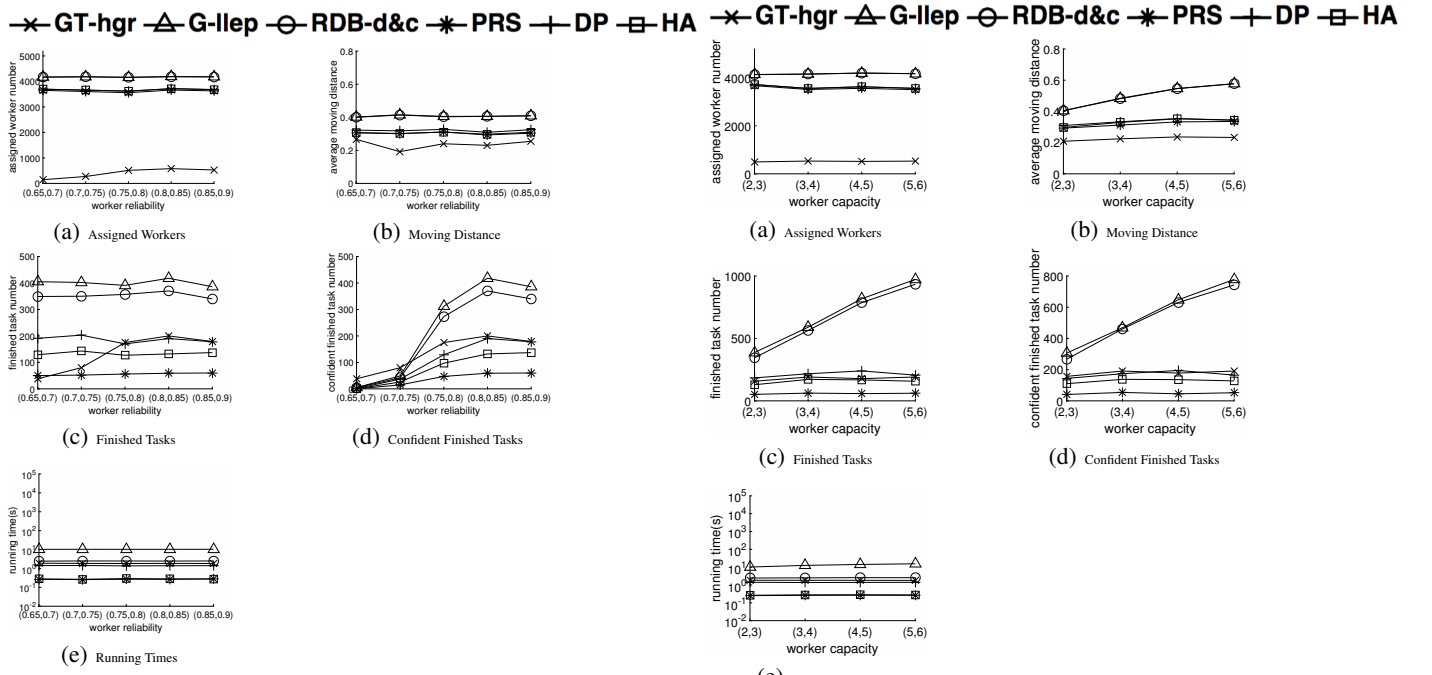


Figure 19: Effects of Workers' Reliabilities r_i (GAUS).

Figure 21: Effects of Workers' Capacities c_j (GAUS).

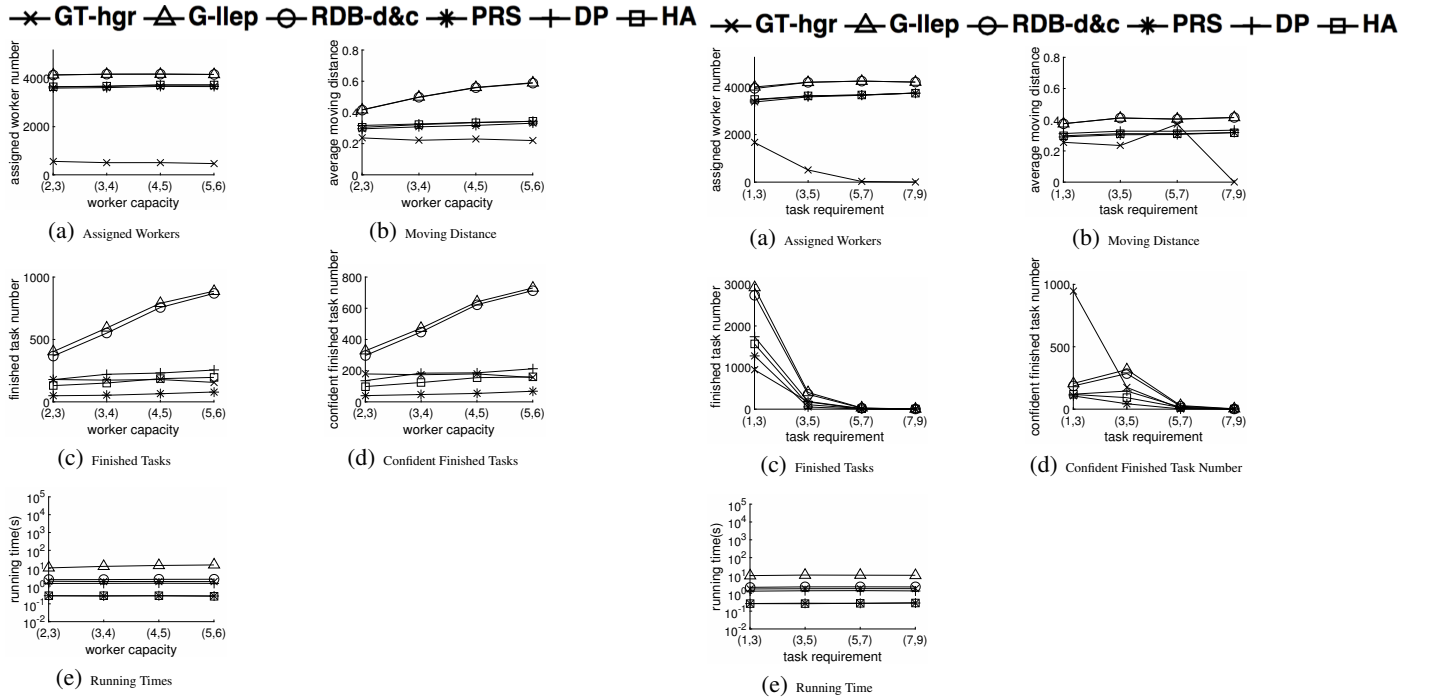


Figure 20: Effects of Workers' Capacities c_j (SKEW).

Figure 22: Effects of Tasks' Required Answer Count a_j (SKEW).

—x— GT-hgr —△— G-llep —○— RDB-d&c —*— PRS —+— DP —□— HA

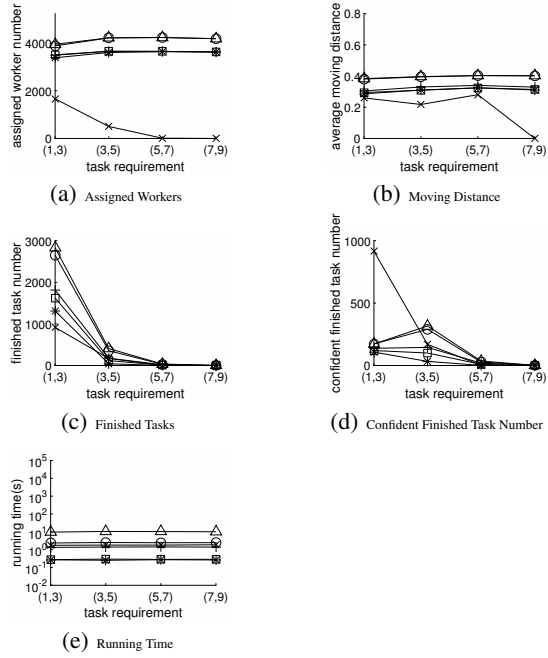


Figure 23: Effects of Tasks' Required Answer Count a_j (GAUS).